UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/705,525	11/10/2003	Attila Barta	RSW920030176US1	5400

46320 7590 07/21/2010

CAREY, RODRIGUEZ, GREENBERG & PAUL, LLP
STEVEN M. GREENBERG
950 PENINSULA CORPORATE CIRCLE
SUITE 2022
BOCA RATON, FL 33487

EXAMINER	
CHEN, QING	
ART UNIT	**PAPER NUMBER**
2191	

MAIL DATE	DELIVERY MODE
07/21/2010	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Application Number: 10/705,525
Filing Date: November 10, 2003
Appellant(s): BARTA ET AL.

Steven M. Greenberg (Reg. No. 44,725)
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed on April 12, 2010 appealing from the Office action mailed on November 9, 2009.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

The Examiner is not aware of any related appeals, interferences, or judicial proceedings which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

(3) Status of Claims

The Examiner agrees with the statement of the status of claims contained in the brief.

(4) Status of Amendments After Final

The Examiner agrees with the statement of the status of amendments contained in the brief.

(5) Summary of Claimed Subject Matter

The Examiner agrees with the summary of claimed subject matter contained in the brief.

(6) Grounds of Rejection to be Reviewed on Appeal

The Examiner agrees with the statement of the grounds of rejection to be reviewed set forth in the brief.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

6,442,754	CURTIS	8-2002
6,675,382	FOSTER	1-2004
6,725,452	TE'ENI et al.	4-2004
6,918,112	BOURKE-DUNPHY et al.	7-2005

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

1. **Claims 1, 3, 4, and 6-13** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,442,754 (hereinafter "Curtis")** in view of **US 6,725,452 (hereinafter "Te'eni")** and **US 6,675,382 (hereinafter "Foster")**.

As per **Claim 1**, Curtis discloses:

- including, in an installation package for the application, a data structure that provides, for each of the plurality of software components from the application, a software component deployment dependency data, an indication of necessary software components for an operation of each of the plurality of software components being installed *(see Figure 5; Column 6: 29-35, "The install program further includes a program object 303 comprised of file set objects 340.*

Within each file set object 340 there are multiple install objects 330. There are several types of install objects--file object 331, registry object 332 ..."; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.");

- loading the installation package into a memory connected to a computer *(see Figure 1: 10; Column 5: 29-31, "The programs in memory 12 includes an operating system (OS) 16 program and application programs, such as an install program 17 or an installer tool kit.")*; and

- using the computer so configured by the installation package *(see Figure 1: 10; Column 5: 29-31, "The programs in memory 12 includes an operating system (OS) 16 program and application programs, such as an install program 17 or an installer tool kit.")*, performing the steps of:

- determining a first plurality of software components previously installed on a system *(see Column 11: 11-20, "... a call to the check_dependency function ... This function*

determines whether the file, program or registry object indicated in the dependency object 400 is installed on the computer.");

- determining a second plurality of software components to be installed on the system *(see Figure 2: 340; Column 11: 23-24, "... a file set 340 is installed.")*;

- accessing a third plurality of software component deployment dependency data *(see Column 13: 18-21, "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate.")*; and

- accessing a sixth plurality of metadata from the data structure regarding the second plurality of software components to be installed and accessing a seventh plurality of metadata regarding the first plurality of software components previously installed *(see Column 13: 13-15 and 21-24, "A root directory includes a sub-directory for each installed program, indicating the program name and version." and "The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends.")*.

However, Curtis does not disclose:

- an indication of incompatibility with a previously installed software component;

- determining a fourth plurality of software components suitable for parallel installation;

- determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations;

- analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system;

- wherein a pre-deployment analysis allows the second plurality of software components to be installed in parallel and in a sequence of groups; and

- wherein an installation time for the application is reduced.

Te'eni discloses:

- an indication of incompatibility with a previously installed software component *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-17, "Virtual upgrade module 36 creates upgrade processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module ...")*; and

- analyzing a sixth plurality of metadata to determine an eight plurality of potential conflicts between a second plurality of software components to be installed and a first plurality of software components previously installed on a system *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-20, "Virtual upgrade module 36 creates upgrade processes for the*

following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module, (b) to activate conflict resolver module 40 in order to check for potential dependency conflicts and to resolve any dependency conflicts that might arise as a result of the planned installation process ...").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Te'eni into the teaching of Curtis to modify Curtis' invention to include an indication of incompatibility with a previously installed software component; and analyzing the sixth plurality of metadata to determine an eight plurality of potential conflicts between the second plurality of software components to be installed and the first plurality of software components previously installed on the system. The modification would be obvious because one of ordinary skill in the art would be motivated to prevent any unsuccessful installation *(see Te'eni – Column 1: 64-66).*

Foster discloses:

- determining a fourth plurality of software components suitable for parallel installation *(see Column 10: 6-8, "... other packages may be concurrently installed that require the presence of package 200 on the system.")*;

- determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations *(see Column 10: 8-10, "... the system checks the dependencies between package 200 and the packages that are being simultaneously installed.")*;

- wherein a pre-deployment analysis allows a second plurality of software components to be installed in parallel and in a sequence of groups *(see Column 10: 8-10, "... the packages that are being simultaneously installed.")*; and

- wherein an installation time for an application is reduced *(see Column 10: 8-10, "... the packages that are being simultaneously installed.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Foster into the teaching of Curtis to modify Curtis' invention to include determining a fourth plurality of software components suitable for parallel installation; determining an order in which the fourth plurality of software components can be grouped for a fifth plurality of parallel installations; wherein a pre-deployment analysis allows the second plurality of software components to be installed in parallel and in a sequence of groups; and wherein an installation time for the application is reduced. The modification would be obvious because one of ordinary skill in the art would be motivated to provide an efficient and simple solution for packaging, distributing and installing software *(see Foster – Column 1: 43-44)*.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- updating the data structure with an identity of a ninth plurality of recently installed software components *(see Column 13: 28-30, "The information in this directory is created whenever a component is installed. For instance, whenever a program is installed, a subdirectory is created under the root directory.")*.

As per **Claim 4**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- providing a user with a plurality of options for the eight plurality of potential conflicts *(see Column 12: 35-45, "If so, control transfers to block 534 where the program displays on the display means 14 the name of the dependent component that was not located on the system and a radio button to allow the user to selectively cause the execution of the install script in the Install 416 or SInstall 418 fields. If there is not install script, then control transfers to block 536 where the program displays information maintained in the install information field 426 to inform the user on where to obtain the dependent component that is needed before the program may be installed.").*

As per **Claim 6**, the rejection of **Claim 4** is incorporated; and Curtis further discloses:

- wherein a second option includes continuing an installation *(see Column 12: 51-53, "When the user selects to install the file, the install program 17 will execute the install script in either the Install 416 or SInstall field 418.").*

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Curtis further discloses:

- upon the exercise of the second option, recording an entry in a log indicative of a conflict and of a continuation of installation *(see Figure 2: 140; Column 7: 4-5, "During install, the log 140 and 'uninstall.Java1' 150 information are built."; Column 8: 24-29, "... providing various logs, e.g. a log for keeping track of what is being installed, and a log that reports the progress of install. Logs are used for both the install and uninstall process. Furthermore, these*

logs are human readable which allows them to be checked, e.g., after a silent install, to ensure that a file has installed successfully.").

As per **Claim 8**, the rejection of **Claim 1** is incorporated; and Curtis further discloses:

- initiating a removal of a software component from a system *(see Figure 6: 560; Column 13: 50-51, "... the program processes a request to uninstall a program.")*; and

- identifying a tenth plurality of remaining software components which depend on the software component to be removed *(see Column 13: 59-62, "The uninstall program may navigate the directory structure from the dependency directory shown in FIG. 5 to determine dependant programs that depend on the program subject to the uninstallation.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and Curtis further discloses:

- providing a user with a plurality of options if the tenth plurality of dependent remaining software components are identified *(see Column 12: 35-45, "If so, control transfers to block 534 where the program displays on the display means 14 the name of the dependent component that was not located on the system and a radio button to allow the user to selectively cause the execution of the install script in the Install 416 or SInstall 418 fields. If there is not install script, then control transfers to block 536 where the program displays information maintained in the install information field 426 to inform the user on where to obtain the dependent component that is needed before the program may be installed.").*

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Curtis further discloses:

- wherein a first option includes aborting a removal *(see Figure 6: 570; Column 13: 62-63, "Control then transfers to block 570 to exit uninstallation ...")*.

As per **Claim 11**, the rejection of **Claim 9** is incorporated; and Curtis further discloses:

- wherein a second option includes continuing a removal *(see Figure 6: 568; Column 13: 55-56, "Otherwise, control transfers to block 568 to proceed with the uninstallation.")*.

As per **Claim 12**, the rejection of **Claim 8** is incorporated; and Curtis further discloses:

- identifying a first software component previously installed on a system which is dependent upon a removed software component *(see Column 13: 4-6 and 64-67, "... before uninstalling a program, a determination may be made as to whether other installed components depend on the file being uninstalled." and "... information indicating the depending programs that should be uninstalled before continuing with the uninstallation of the program, which is a dependent program.")*; and

- determining an identity of a second software component upon which the first software component depends *(see Column 13: 1-4, "During installation of a dependent program, dependency information from the Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file.")*.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Curtis further discloses:

- installing the second software component upon which the first software component depends *(see Column 13: 1-4, "During installation of a dependent program, dependency*

information from the Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file."); and

- creating a dependency link between the first software component and the second software component *(see Column 13: 1-4, "Dependency Object 400 may be written to a dependency location indicating dependent components of the installed file.")*.

2. **Claim 5** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Curtis** in view of **Te'eni** and **Foster** as applied to Claim 4 above, and further in view of **US 6,918,112 (hereinafter "Bourke-Dunphy")**.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Curtis, Te'eni, and Foster do not disclose:

- wherein a first option includes aborting an installation.

Bourke-Dunphy discloses:

- wherein a first option includes aborting an installation *(see Figure 5: 236; Column 8: 35-38, "... the user may select a CANCEL action button 236 to return to the component selection user interface ... where the user may manually modify the component selections.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bourke-Dunphy into the teaching of Curtis to modify Curtis' invention to include wherein a first option includes aborting an installation. The modification would be obvious because one of ordinary skill in the art would be motivated to

allow the user to exit the current installation, correct the error identified, and reinitiate the installation procedure *(see Bourke-Dunphy – Column 1: 27-34)*.

3. **Claim 41** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Curtis** in view of **Foster**.

As per **Claim 41**, Curtis discloses:

- accessing the semantic model to obtain a dependency information about software components of an application *(see Figure 5; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.")*;

- including a semantic model in an installation package of the application *(see Figure 5; Column 6: 29-35, "The install program further includes a program object 303 comprised of file set objects 340. Within each file set object 340 there are multiple install objects 330. There*

are several types of install objects--file object 331, registry object 332 ..."; Column 13: 7-27, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree.");

- responsive to loading the installation package into a memory connected to a computer, using the computer so configured by the installation package *(see Figure 1: 10; Column 5: 29-31, "The programs in memory 12 includes an operating system (OS) 16 program and application programs, such as an install program 17 or an installer tool kit.")* to perform steps comprising:

- storing a first record of each of a plurality of the software components that is to be deployed in a read file *(see Column 9: 20-24, "The dependency object 400 is used during install and uninstall operations to determine whether any files or programs upon which the program to install needs in order to operate are installed and determine whether any programs to uninstall are required by already installed programs.")*;

- storing a second record of each of a plurality of previously installed software components in a registry file *(see Column 13: 7-27, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree.")*;

- when the read file is available to deploy, examining the registry file and accessing the semantic model to obtain a plurality of dependency information indicating a plurality of relationships among the plurality of the software components to be installed in the target and among a plurality of previously installed software components *(see Column 13: 7-47, "Each installed program includes a next level subdirectory for each file set of the installed program, indicating the file set name and version. Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate." and "The registry route 436 in the dependency object 400 indicates the location of the dependency directory in the registry file where the dependency information for a particular dependency object is maintained." and "The registry route 436 is used to determine the dependency directory of where to write dependency information when the program is installed.")*;

- using the plurality of dependency information to group the plurality of the software components into sets of software components with like dependency levels, wherein a first set of software components from amongst the sets of software components has no dependencies, a second set of software components from amongst the sets of software components has dependencies only on the first set of software components, and a third set of software components from amongst the sets of software components has dependencies only on the first and second sets of software components *(see Figure 5; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets,*

and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values.");

- when a component is installed, updating the registry file *(see Column 11: 63-65, "As discussed, information on installed components is added to a registry file, e.g., registry 220, when the components are installed.")*;

- when a conflict is identified, taking an appropriate action *(see Column 12: 35-45, "If so, control transfers to block 534 where the program displays on the display means 14 the name of the dependent component that was not located on the system and a radio button to allow the user to selectively cause the execution of the install script in the Install 416 or SInstall 418 fields. If there is not install script, then control transfers to block 536 where the program displays information maintained in the install information field 426 to inform the user on where to obtain the dependent component that is needed before the program may be installed.")*; and

- displaying a progress report by labeling the plurality of the software components in the semantic model in a selected level of granularity *(see Figure 2: 140; Column 7: 4-5, "During install, the log 140 and 'uninstall.Java1' 150 information are built."; Column 8: 24-29, "... providing various logs, e.g. a log for keeping track of what is being installed, and a log that*

reports the progress of install. Logs are used for both the install and uninstall process. Furthermore, these logs are human readable which allows them to be checked, e.g., after a silent install, to ensure that a file has installed successfully.").

However, Curtis does not disclose:

- installing the first set of software components in parallel;

- responsive to completing installation of the first set of software components, installing the second set of software components in parallel; and

- responsive to completing installation of the second set of software components, installing the third set of software components in parallel.

Foster discloses:

- installing a first set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.")*;

- responsive to completing installation of the first set of software components, installing a second set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.")*; and

- responsive to completing installation of the second set of software components, installing a third set of software components in parallel *(see Column 10: 6-10, "... other packages may be concurrently installed that require the presence of package 200 on the system." and "... the packages that are being simultaneously installed.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Foster into the teaching of Curtis to modify Curtis' invention to include installing the first set of software components in parallel; responsive to completing installation of the first set of software components, installing the second set of software components in parallel; and responsive to completing installation of the second set of software components, installing the third set of software components in parallel. The modification would be obvious because one of ordinary skill in the art would be motivated to provide an efficient and simple solution for packaging, distributing and installing software *(see Foster – Column 1: 43-44)*.

(10) Response to Argument

THE REJECTION OF CLAIMS 1, 3, 4 AND 6 THROUGH 13 UNDER 35 U.S.C. § 103(A)

In the Appeal Brief, Appellant argues:

a) Integral to claim 1 is the inclusion within the installation package for each software component of the application, of an indication of incompatibility with a previously installed software component. This teaching cannot be found in Te'eni.

Appellants maintained as much in the Seventh Amendment. Specifically, Appellants argued on page 13 of the Seventh Amendment:

> In referring to column 1, lines 61 through 64 of Te'eni, it is clear that only a general statement exists that conflicts arise during installation of components. Column 5, lines 10 through 17 only add that an upgrade process is created for a

> sequence of tasks that includes collecting information for a "dependency conflicts resolving process".
>
> In response, Examiner stated on page 19 of the Final Office Action:
>
> Note that it is well-known in the art that when performing an upgrade of a software program, dependency conflicts may arise among the software components present (previously installed software components) and the software components to be installed. Te'eni's invention attempts to resolve such dependency conflicts by utilizing a virtual upgrade module to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process (an indication of incompatibility with a previously installed software component).
>
> Of import, while at present it is known that when performing an upgrade of a software program, dependency conflicts may arise, Examiner has provided no evidence that such knowledge was present in the art at the time of Appellants' invention. Further, and more importantly, Appellants' claims require much more than mere knowledge that a conflict may have arisen. Rather, Appellants' claims require in accordance with the plain claim language of claim 1 "a data structure that provides, for each of the plurality of software components from the application, … an indication of incompatibility with a previously installed software component." Examiner has not attempted to map each claim term set forth above to Te'eni as required by M.P.E.P. 2141.
>
> *(See Appeal Brief – page 11 to page 12, emphasis in original.)*

Examiner's response:

a) Examiner disagrees. Appellant's arguments are not persuasive for at least the following reasons:

First, with respect to the Appellant's assertion that the Examiner has provided no evidence that knowledge of dependency conflict may arise during software upgrade was present in the art at the time of the Appellant's invention, the Examiner respectfully submits that the "Background of the Invention" section of Te'eni's disclosure clearly discloses that "[w]hen performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed *(see Column 1: 61-64)*." Furthermore, note that Te'eni's invention was filed on June 1, 2000, which is more than three years before the filling of the Appellant's invention on November 10, 2003. Thus, as can be seen, the knowledge of dependency conflict may arise during software upgrade is clearly present in the art and known to those of ordinary skill in the art at the time the Appellant's invention was made.

Second, the Examiner would like to point out that Curtis clearly discloses "including, in an installation package for the application, a data structure that provides, for each of the plurality of software components from the application, a software component deployment dependency data, an indication of necessary software components for an operation of each of the plurality of software components being installed" *(see Figure 5; Column 6: 29-35, "The install program further includes a program object 303 comprised of file set objects 340. Within each file set object 340 there are multiple install objects 330. There are several types of install objects--file object 331, registry object 332 ..."; Column 13: 7-27 and 33-37, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components*

in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values."). Note that Curtis' invention describes using a data structure to store information about installed programs and dependent components on which the installed programs depend. However, Curtis does not disclose "an indication of incompatibility with a previously installed software component." Examiner relied upon Te'eni for its specific teaching of "an indication of incompatibility with a previously installed software component."

Third, with respect to the Appellant's assertion that Te'eni does not teach "an indication of incompatibility with a previously installed software component," as previously pointed out in the Non-Final Rejection (mailed on 04/03/2009) and the Final Rejection (mailed on 11/09/2009) and further clarified hereinafter, the Examiner respectfully submits that Te'eni clearly discloses "an indication of incompatibility with a previously installed software component" *(see Column 1: 61-64, "When performing the predefined procedures necessary for an upgrade to be implemented frequently dependency conflicts may arise among the components present and the components to be installed."; Column 5: 10-17, "Virtual upgrade module 36 creates upgrade processes for the following tasks that are performed sequentially: (a) to collect all the information necessary for the dependency analysis and the dependency conflicts resolving*

process such as the relevant component data information units from component data table 28, the encoded dependency rules from xor-rules table 32 and from add-remove rules table 34 module ..."). Note that it is well-know in the art that when performing an upgrade of a software program, dependency conflicts may arise among the software components present (previously installed software components) and the software components to be installed. Te'eni's invention attempts to resolve such dependency conflicts by utilizing a virtual upgrade module to collect all the information necessary for the dependency analysis and the dependency conflicts resolving process (an indication of incompatibility with a previously installed software component).

Therefore, for at least the reasons set forth above, the rejection made under 35 U.S.C. § 103(a) with respect to Claim 1 is proper and therefore, maintained.

THE REJECTION OF CLAIM 41 UNDER 35 U.S.C. § 103(A)

In the Appeal Brief, Appellant argues:

b) As noted on page 15 of the Seventh Amendment, column 13, lines 7 through 27 and 33 through 37 of Curtis lacks the important teaching of using dependency information in order to group software components into sets according to like dependency levels. In response, on pages 20 and 21 of the Final Office Action Examiner addressed much of the rejection of claim 41, however, the Examiner failed to address Appellants' arguments in regard to "grouping" software components into sets.

(See Appeal Brief – page 13 to page 14.)

Examiner's response:

b) Examiner disagrees. With respect to the Appellant's assertion that the Examiner failed to address the Appellant's arguments in regard to "grouping" software components into sets, as previously pointed out in the Non-Final Rejection (mailed on 04/03/2009) and the Final Rejection (mailed on 11/09/2009) and further clarified hereinafter, the Examiner respectfully submits that Curtis clearly discloses using dependency information in order to group software components into sets according to like dependency levels *(see Figure 5; Column 13: 7-27 and 33-47, "... a data structure ... is maintained in the registry object or registry database 220, indicating installed programs and dependent components on which installed programs depend. In the embodiment of FIG. 5, the data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree." and "Each installed file set component has a Dependency subdirectory which includes information on each dependent component on which the file set and program depend in order to operate. The dependency subdirectory would list the program name, version, fileset name, and fileset version for each program on which the fileset including the dependency subdirectory depends." and "... the dependency directory may indicate dependent file sets or registry objects that are the subject matter of the processed dependency object. If there are no dependent components, then the dependency directory will contain no values." and "During installation, after the program is installed and all dependency objects are processed and the Passed field 428 is "on," the program may indicate values in the Dependency directory by processing all the passed dependency objects in the installed file set and writing the program name and version indicated in the dependency object fields 408 and 410 to the dependency directory.")*. Attention is drawn to

Figure 5 of Curtis which clearly illustrates a data structure indicating installed programs and dependent components on which the installed programs depend. The data structure is a hierarchical arrangement of programs, file sets, and dependent components in the form of a directory tree. Note that a dependency subdirectory of the directory tree includes information (dependency information) on each dependent component on which the file set and program depend in order to operate. The dependent components on which the file set and program depend in order to operate are processed and their names and versions are written to the dependency subdirectory of the file set and program. Thus, one of ordinary skill in the art would readily comprehend that the set of dependent components on which the file set and program depend in order to operate are grouped in the same dependency subdirectory (like dependency level) of the file set and program.

Therefore, for at least the reason set forth above, the rejection made under 35 U.S.C. § 103(a) with respect to Claim 41 is proper and therefore, maintained.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the Examiner in the Related Appeals and Interferences section of this Examiner's answer.

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted,

Qing Chen

/Q. C./

Examiner, Art Unit 2191

Conferees:

/Wei Y Zhen/

Supervisory Patent Examiner, Art Unit 2191

/Lewis A. Bullock, Jr./

Supervisory Patent Examiner, Art Unit 2193